When Do Bounds and Domain Propagation Lead to the Same Search Space

Christian Schulte
Programming Systems Lab
Universität des Saarlandes, Germany
schulte@ps.uni-sb.de

Peter J. Stuckey
Dept. of Comp. Sci. & Soft. Eng.
University of Melbourne, Australia
pjs@cs.mu.oz.au

ABSTRACT

This paper explores the question of when two propagation-based constraint systems have the same behaviour, in terms of search space. We categorise the behaviour of domain and bounds propagators for primitive constraints, and provide theorems that allow us to determine propagation behaviours for conjunctions of constraints. We then show how we can use this to analyse CLP(FD) programs to determine when we can safely replace domain propagators by more efficient bounds propagators without increasing search space.

Keywords

Constraint (logic) programming, finite domain constraints, bounds propagation, domain propagation, abstract interpretation, program analysis

1. INTRODUCTION

In building a finite domain constraint programming solution to a combinatoric problem a tradeoff arises in the choice of propagation that is used for each constraint: stronger propagation methods are more expensive to execute but may detect failure earlier; weaker propagation methods are (generally) cheaper to execute but may (exponentially) increase the search space explored to find an answer. In this paper we investigate the possibility of analysing finite domain constraint systems, or constraint programs, and determining whether the propagation methods used for some constraints could be replaced by simpler, and more efficient alternatives without increasing the size of the search space.

Consider the following example constraint:

$$[x_1, x_2, x_3, x_4] :: [0..10], x_1 \leq x_2, 2x_2 = 3x_3 + 1, x_3 \leq x_4$$

Each of the constraints could be implemented using domain propagation or bounds propagation. Clearly if each constraint is implemented using domain propagation we have stronger information, and the search space explored in order to find all solutions for the problem will be no larger than if we used bounds propagation. The question we ask is, can we get the same search space with bounds propagation.

Domain propagation on the constraints $x_1 \leq x_2$ and $x_3 \leq x_4$ is equivalent to bounds propagation since the constraints only place upper and lower bounds on their variables. This is not the case for $2x_2 = 3x_3 + 1$ where domain propagation reduces the domains of x_2 to $\{2, 5, 8\}$ and x_3 to $\{1, 3, 5\}$, while bounds propagation reduces x_2 to $[2 .. 8]$ and x_3 to $[1 .. 5]$. The question is: will execution require more search, if we use bounds propagation for this constraint as well?

Suppose that we use a labelling strategy that either sets a variable to its lower bound, or constrains it to be greater than its lower bound. Then none of the constraints added during search creates holes in the domains. (This is in contrast to a strategy where we set a variable to equal its middle value, or to exclude its middle value.) Hence the only holes in the domains of x_2 and x_3 will come from the constraint $2x_2 = 3x_3 + 1$. We will show that if the domains of x_2 and x_3 have no holes, domain propagation on $2x_2 = 3x_3 + 1$ fails iff bounds propagation fails. Hence the search space is the same for both bounds and domain propagation.

While for this simple example the advantage of bounds propagation over domain propagation may not seem significant, for more complex constraints there can be significant differences in efficiency of domain and bounds propagation. For example, domain propagation for `alldifferent` on n variables is $O(n^{2.5})$ [16], while bounds propagation is $O(n \log n)$ [15] and even $O(n)$ in common cases [13]. Similarly, domain propagation for a linear equation on n variables is exponential while bounds propagation is $O(n)$.

In this paper we investigate when bounds and domain propagation will lead to the same search space.

The contributions of this paper are:

- We classify the behaviour of propagators for common primitive constraints, in particular introducing the crucial notion of *endpoint-relevant* propagators.
- We give theorems that allow us to extend reasoning about propagators for a single constraint to reasoning about propagators for a conjunction of constraints.
- We define an analysis algorithm for CLP(FD) programs that determines where we can replace domain propagators with bounds propagators without increasing the search space.
- We show examples where our analysis detects search space equivalent replacements and show the possible performance benefits that arise.

PPDP 2001 Firenze, Italy

Previous authors [13, 15] have noted the difference in efficiency in bounds and domain propagation, for particular primitive constraints but not considered when different propagators lead to the same search space. The closest related work is that of Harvey and Stuckey [7], who consider the relative propagation strengths of different equivalent forms of constraints. Although they consider both domain and bounds propagation, they never compare bounds propagation to domain propagation.

While there has been considerable success in optimizing constraint programs over real linear constraints [8], there has been little progress in optimizing finite domain CLP programs. Much of this stems from the difficulty in effectively analysing the behaviour of CLP(FD) solvers. In this paper we have made a first step in this direction.

The remainder of the paper is organized as follows: in the next section we introduce terminology and define domain and bounds propagators. We then investigate properties of propagators and sets of propagators that allow us to prove search space equivalence. In Section 3, we define an analysis of CLP(FD) programs to gather information about propagation. We use this to define a program transformation that annotates individual constraints with the form of propagation we should use for them. Finally in Section 4 we conclude and give some directions for extending the work.

2. PROPAGATION BASED SOLVING

2.1 Basic Definitions

In this paper we consider integer constraint solving. A *primitive linear constraint* c is an equality ($=$), inequality ($\leq$), or disequation ($\neq$), which are written $\sum_{i=1}^{n} a_i x_i$ op d where x_i are integer variables, a_i, d are integers, and op $\in \{=, \leq, \neq\}$. A *primitive nonlinear constraint* is a multiplication $x_1 = x_2 \times x_3$, a squaring $x_1 = x_2 \times x_2$, a positive squaring $x_1 = x_2 \times x_2 \wedge x_2 \geq 0$, an alldifferent constraint `alldifferent`$([x_1, \ldots, x_n])$, or a default constraint `default`$([x_1, \ldots, x_n])$ (representing a nonlinear constraint with no further information on its constraint propagation available). A *constraint* is a conjunction of primitive constraints, which we will sometimes treat as a set of primitive constraints.

We use the notation $[x_1, \ldots, x_n] :: [l \mathinner{..} u]$ as shorthand for the conjunction of inequalities

$$x_1 \geq l, x_1 \leq u, \ldots, x_n \geq l, x_n \leq u$$

An *integer (real) valuation* θ is a mapping of variables to integer (resp. real) values, written $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$. We extend the valuation θ to map expressions and constraints involving the variables in the natural way. Let *vars* be the function that returns the set of (free) variables appearing in a constraint or valuation. A valuation θ is an integer (real) *solution* of a constraint c, if $\mathcal{Z} \models_\theta c$ (resp. $\mathcal{R} \models_\theta c$).

A *domain* D is a complete mapping from a fixed (countable) set of variables $\mathcal{V}$ to finite sets of integers. A *false domain* D is a domain with $D(x) = \emptyset$ for some x. The *intersection* of two domains D_1 and D_2, denoted $D_1 \sqcap D_2$, is defined by the domain $D_3(x) = D_1(x) \cap D_2(x)$ for all x. A domain D_1 is *stronger* than a domain D_2, written $D_1 \sqsubseteq D_2$, if $D_1(x) \subseteq D_2(x)$ for all variables x. A domain D_1 is stronger than (equal to) a domain D_2 w.r.t. variables V, denoted $D_1 \sqsubseteq_V D_2$ (resp. $D_1 =_V D_2$), if $D_1(x) \subseteq D_2(x)$ (resp. $D_1(x) = D_2(x)$) for all $x \in V$.

In an abuse of notation, we define a valuation θ to be an element of a domain D, written $\theta \in D$, if $\theta(x_i) \in D(x_i)$ for all $x_i \in vars(\theta)$. We will be interested in determining the infimums and supremums of expressions with respect to some domain D. Define the *infimum* and *supremum* of an expression e with respect to a domain D as $\inf_D e = \inf\ \{\theta(e) | \theta \in D\}$ and $\sup_D e = \sup\ \{\theta(e) | \theta \in D\}$.

A *propagator* f for variable x is a function mapping a domain D to a set of values representing the possible values for x. A propagator only considers part of the domain corresponding to some subset of variables of interest which we denote by $vars(f)$.

We can extend propagators f for a variable x to map a domain D to another domain D'. Let $prop(f, D)$ denote the extension of f to map domains to domains, defined by $D'(x') = D(x')$ for $x' \neq x$, and $D'(x) = D(x) \cap f(D)$.

A propagator f is correct for a constraint c, iff

$$\{\theta \in D \mid \mathcal{Z} \models_\theta c\} = \{\theta \in prop(f, D) \mid \mathcal{Z} \models_\theta c\}$$

EXAMPLE 2.1. *For the constraint $c \equiv x_1 \geq x_2 + 1$ the function $f(D) = \{d \in D(x_1) \mid d \geq \inf_D x_2 + 1\}$ is a correct propagator for variable x_1. Let $D_1(x_1) = D_1(x_2) = \{1, 5, 8\}$, then $f(D_1) = \{5, 8\}$ and $prop(f, D_1) = D_2$ where $D_2(x_1) = \{5, 8\}$ and $D_2(x_2) = \{1, 5, 8\}$.* □

A *propagation solver* for a set of propagators F and an initial domain D, $solv(F, D)$, repeatedly applies all the propagators in F starting from domain D until there is no further change in resulting domain. In other words, $solv(F, D)$ returns a new domain defined by

$$\begin{aligned} iter(F, D) &= \bigsqcap_{f \in F} prop(f, D) \\ solv(F, D) &= \text{gfp}(\lambda d.iter(F, d))(D). \end{aligned}$$

where gfp denotes the greatest fixpoint w.r.t $\sqsubseteq$ lifted to functions.

A domain D is *domain-consistent* for a constraint c if D is the least domain containing all integer solutions $\theta \in D$ of c, i.e, there does not exist $D' \sqsubset D$ such that $\theta \in D \wedge \mathcal{Z} \models_\theta c \Rightarrow \theta \in D'$.

A set of propagators F maintains *domain-consistency* for a constraint c, if $solv(F, D)$ is always domain-consistent for c.

Define the *domain-consistency propagator* for a constraint c and a variable x, $dom(c, x)$, as follows

$$dom(c, x)(D) = \{\theta(x) \mid \theta \in D \text{ and } \theta \text{ is a solution of } c\}.$$

EXAMPLE 2.2. *Consider the constraint $c \equiv x_1 = 3x_2 + 5x_3$ and the domain $D(x_1) = \{2, 3, 4, 5, 6, 7\}$, $D(x_2) = \{0, 1, 2\}$, and $D(x_3) = \{-1, 0, 1, 2\}$. The solutions of c are $\theta_1 = \{x_1 \mapsto 3, x_2 \mapsto 1, x_3 \mapsto 0\}$, $\theta_2 = \{x_1 \mapsto 5, x_2 \mapsto 0, x_3 \mapsto 1\}$, and $\theta_3 = \{x_1 \mapsto 6, x_2 \mapsto 2, x_3 \mapsto 0\}$. Hence, $iter(\{dom(c, x_1), dom(c, x_2), dom(c, x_3)\}, D)$ gives a domain D' such that $D'(x_1) = \{3, 5, 6\}$, $D'(x_2) = \{0, 1, 2\}$, and $D'(x_3) = \{0, 1\}$. D' is domain-consistent with respect to c, hence also $solv(\{dom(c, x_1), dom(c, x_2), dom(c, x_3)\}, D) = D'$.* □

A *range* of integers $[l \mathinner{..} u]$ is the set of integers $\{l, l + 1, \ldots, u\}$, or $\emptyset$ if $l > u$. A domain is a *range domain* if $D(x_i)$ is a range for all x_i. Let $D' = range(D)$ be the

smallest range domain containing D, i.e. domain $D'(x_i) = [\inf_D x_i \,..\, \sup_D x_i]$. A domain D_1 is *bounds-stronger* than a domain D_2, written $D_1 \stackrel{b}{\sqsubseteq} D_2$, either D_1 is a false domain or $range(D_1) \sqsubseteq range(D_2)$. Two domains D_1 and D_2 are *bounds-equal*, denoted $D_1 \stackrel{b}{\equiv} D_2$, if $range(D_1) = range(D_2)$.

A domain D is *bounds-consistent* for a constraint c and a variable x_i with $vars(c) = \{x_1, \ldots, x_n\}$, if for each $d_i \in \{\inf_D x_i, \sup_D x_i\}$ there exist *real numbers* d_j with $\inf_D x_j \leq d_j \leq \sup_D x_j$, $1 \leq j \neq i \leq n$ such that $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$ is a *real* solution of c. A domain D is *bounds-consistent* for a constraint c, if it is bounds-consistent for c and each $x \in vars(c)$.

A set of propagators F maintains *bounds-consistency* for a constraint c, if $solv(F, D)$ is always bounds-consistent for c.

Define the *bounds-consistency propagator* for a constraint c and variable x, $bnd(c, x)$, as in Figure 1.

EXAMPLE 2.3. *Consider the same constraint c and domain D as in Example 2.2. Calculation of*

$$D' = iter(\{bnd(c, x_1), bnd(c, x_2), bnd(c, x_3)\}, D)$$

determines that

$$\begin{array}{rcl} D'(x_1) & = & [l_1 \,..\, u_1] = [2 \,..\, 7] \\ & & \quad with \quad l_1 = \sup\left\{\left\lceil \frac{0+3\times 0+5\times -1}{1} \right\rceil, 2\right\} \\ & & \quad and \quad u_1 = \inf\left\{\left\lfloor \frac{0+3\times 2+5\times 2}{1} \right\rfloor, 7\right\} \end{array}$$

and

$$\begin{array}{rcl} D'(x_3) & = & [l_3 \,..\, u_3] = [0 \,..\, 1] \\ & & \quad with \quad l_3 = \sup\left\{\left\lceil \frac{0-3\times 2+1\times 2}{5} \right\rceil, -1\right\} \\ & & \quad and \quad u_3 = \inf\left\{\left\lfloor \frac{0-3\times 0+1\times 7}{5} \right\rfloor, 2\right\} \end{array}$$

While the domain of x_3 is modified, the domains of x_1 and x_2 remain unchanged. The resulting domain D' is bounds-consistent with the constraint c. Notice that bounds propagation has determined less information than domain propagation. □

Note that common definitions for bounds propagators for $x_1 = x_2 \times x_3$ (see for example [11]) do not maintain bounds-consistency. For the domain $D(x_1) = [4 \,..\, 8]$, $D(x_2) = [-1 \,..\, 3]$, $D(x_3) = [-1 \,..\, 3]$, the definition of [11] gives no propagation, but $x_2 = -1$ has no solution. It is relatively straightforward to prove that the propagators defined in Figure 1 maintain bounds-consistency.

THEOREM 2.4. *For all c, the set of propagators*

$$\{bnd(c, x) \mid x \in vars(c)\}$$

defined in Figure 1 maintains bounds-consistency for c. □

We conclude this section by briefly defining CLP(FD) programs. For more information see e.g. [11, 17].

An *atom* is of the form $p(x_1, \ldots, x_n)$ where p is a *predicate symbol* and $x_1, \ldots, x_n$ are distinct variables in $\mathcal{V}$. A *literal* is an atom, labelling literal or primitive constraint. A *goal* is a sequence of literals. A *CLP(FD) program* is a set of rules A `:-` G where A is an atom, and G is a goal. Note that we assume here a restricted form for programs, all atoms appear with distinct variable arguments. It is easy to translate any program to an equivalent program of this form.

A *labelling literal* is (for our purposes) one of

`labelling`$([x_1, \ldots, x_n])$
`labellingff`$([x_1, \ldots, x_n])$
`labellingmid`$([x_1, \ldots, x_n])$

where $x_1, \ldots, x_n$ are distinct variables. The role of a labelling literal is to ensure that every variable involved eventually takes a fixed value. There are many kinds of labelling possible, but the three we use `labelling` (default), `labellingff` (first-fail labelling) and `labellingmid` (middle-out value ordering) illustrate the three different kinds of propagation behaviour. `labelling`, and other labellings (such as labelling the variable with least minimum value) only depend on the endpoints of a domain and only add inequality constraints. `labellingff` calculates which variable x to label using the size of the domain $|D(x)|$, hence it depends on the entire domain, but it only adds inequality constraints. `labellingmid` and other labellings not only depend on the entire domain but also add disequality constraints in the disjunction. See [11] for more details.

2.2 Categorising Propagators

In order to reason about the propagation behaviour of propagators corresponding to primitive constraints, we need to be able to categorise their behaviour. In order for bounds propagation to be as powerful as domain propagation we will need to understand how individual propagators relate to bounds.

DEFINITION 2.5. *A propagator f is* bounds-only*, if $f(D)$ is a range for all domains D.*

A propagator f is bounds-preserving*, if for all domains D such that $D(x)$ is a range for all $x \in vars(f)$, then $f(D)$ is a range.*

EXAMPLE 2.6. *Clearly all bounds propagators are bounds-only and thus also bounds-preserving.*

Typically domain propagators are not bounds-preserving, for example $dom(x_1 = 2x_2, x_1)$ is not bounds-preserving. Some domain propagators are however bounds-preserving, for example $dom(x_1 = 2x_2, x_2)$, or $dom(x_1 = x_2 + 3, x_1)$ as well as $dom(x_1 = x_2 + 3, x_2)$. □

EXAMPLE 2.7. *Note that propagation is highly dependent on the nature of the constraints. For example, if $c_1 \equiv x_1 \geq 3x_2$ and $c_2 \equiv x_1 \leq 3x_2+1$, then $dom(c_1, x_1)$ and $dom(c_2, x_1)$ are both bounds-only. But domain propagation on x_1 for the combined constraint $c_1 \wedge c_2$ is not bounds-only. For example, if $D(x_1) = D(x_2) = [0 \,..\, 8]$ and $D' = prop(dom(c_1 \wedge c_2, x_1), D)$ we have that $D'(x_1) = \{0, 1, 3, 4, 6, 7\}$.* □

In order to replace one set of propagators by another we need to have notions of equivalence between sets of propagators.

DEFINITION 2.8. *Two sets of propagators F_1 and F_2 are* equivalent*, if for each domain D, $solv(F_1, D) = solv(F_2, D)$.*

Two sets of propagators F_1, F_2 are bounds-equivalent*, iff for each domain D, $solv(F_1, D) \stackrel{b}{\equiv} solv(F_2, D)$.*

Equivalent sets of propagators of course can be used to replace each other in any context. Clearly a bounds propagator and a domain propagator will rarely be equivalent, since the domain propagator will remove values from inside domains. Hence we introduce bounds-equivalence.

The key to ensuring that two sets of propagators lead to the same search space is the following obvious result.

- if $c \equiv \sum_{i=1}^{n} a_i x_i = d$,

$$bnd(c, x_j)(D) = \left[\left\lceil \inf_D \left(\frac{d - \sum_{i=1, i \neq j}^{n} a_i x_i}{a_j} \right) \right\rceil \ .. \ \left\lfloor \sup_D \left(\frac{d - \sum_{i=1, i \neq j}^{n} a_i x_i}{a_j} \right) \right\rfloor \right]$$

- if $c \equiv \sum_{i=1}^{n} a_i x_i \leq d$, $a_j > 0$,

$$bnd(c, x_j)(D) = \left[\inf_D x_j \ .. \ \left\lfloor \frac{d - \sum_{i=1, i \neq j}^{n} \inf_D(a_i x_i)}{a_j} \right\rfloor \right]$$

- if $c \equiv \sum_{i=1}^{n} a_i x_i \leq d$, $a_j < 0$,

$$bnd(c, x_j)(D) = \left[\left\lceil \frac{d - \sum_{i=1, i \neq j}^{n} \inf_D(a_i x_i)}{a_j} \right\rceil \ .. \ \sup_D x_j \right]$$

- if $c \equiv \sum_{i=1}^{n} a_i x_i \neq d$,

$$bnd(c, x_j)(D) = \begin{cases} [\inf_D x_j + 1 \ .. \ \sup_D x_j] & \inf_D x_j = \frac{d - \sum_{i=1, i \neq j}^{n} a_i d_i}{a_j} \text{ where } D(x_i) = \{d_i\} \\ [\inf_D x_j \ .. \ \sup_D x_j - 1] & \sup_D x_j = \frac{d - \sum_{i=1, i \neq j}^{n} a_i d_i}{a_j} \text{ where } D(x_i) = \{d_i\}. \\ [\inf_D x_j \ .. \ \sup_D x_j] & \text{otherwise} \end{cases}$$

- if $c \equiv x_1 = x_2 \times x_3$,

$$bnd(c, x_1)(D) = [\inf E \ .. \ \sup E]$$
where $E = \{\inf_D x_2 \times \inf_D x_3, \inf_D x_2 \times \sup_D x_3, \sup_D x_2 \times \inf_D x_3, \sup_D x_2 \times \sup_D x_3\}$

$$bnd(c, x_2)(D) = \begin{cases} [\inf_D x_2 \ .. \ \sup_D x_2] & \inf_D x_3 = \sup_D x_3 = 0 \wedge 0 \in [\inf_D x_1 \ .. \ \sup_D x_1] \\ \emptyset & \inf_D x_3 = \sup_D x_3 = 0 \wedge 0 \notin [\inf_D x_1 \ .. \ \sup_D x_1] \\ [\lceil \inf E \rceil \ .. \ \lfloor \sup E \rfloor] & \inf_D x_3 > 0 \vee \sup_D x_3 < 0 \\ [\inf R \ .. \ \sup R] & \text{otherwise} \end{cases}$$
where $E = \{\inf_D x_1 / \inf_D x_3, \inf_D x_1 / \sup_D x_3, \sup_D x_1 / \inf_D x_3, \sup_D x_1 / \sup_D x_3\}$
and $D_1(x_1) = D(x_1), D_1(x_2) = D(x_2), D_1(x_3) = [1 \ .. \ \sup_D x_3]$
$D_2(x_1) = D(x_1), D_2(x_2) = D(x_2), D_2(x_3) = [0 \ .. \ 0]$
$D_3(x_1) = D(x_1), D_3(x_2) = D(x_2), D_3(x_3) = [\inf_D x_3 \ .. \ -1]$
$R = \cup_{m \in \{1,2,3\}} (bnd(c, x_2)(D_m) \cap [\inf_D x_2 \ .. \ \sup_D x_2])$

The propagator for $bnd(c, x_3)$ is defined analogously to $bnd(c, x_2)$.

- if $c \equiv x_1 = x_2 \times x_2$,

$$bnd(c, x_1)(D) = \begin{cases} \left[(\inf_D x_2)^2 \ .. \ (\sup_D x_2)^2 \right] & \inf_D x_2 \geq 0 \\ \left[(\sup_D x_2)^2 \ .. \ (\inf_D x_2)^2 \right] & \sup_D x_2 \leq 0 \\ \left[0 \ .. \ \sup\{(\inf_D x_2)^2, (\sup_D x_2)^2\} \right] & \text{otherwise} \end{cases}$$

$$bnd(c, x_2)(D) = \begin{cases} \left[\lceil \sqrt{\inf_D x_1} \rceil \ .. \ \lfloor \sqrt{\sup_D x_1} \rfloor \right] & \inf_D x_2 \geq 0 \\ \left[\lceil -\sqrt{\sup_D x_1} \rceil \ .. \ \lfloor -\sqrt{\inf_D x_1} \rfloor \right] & \sup_D x_2 \leq 0 \\ [\inf R \ .. \ \sup R] & \text{otherwise} \end{cases}$$
where $D_1(x_1) = D(x_1), D_1(x_2) = [0 \ .. \ \sup_D x_2]$
$D_2(x_1) = D(x_1), D_2(x_2) = [\inf_D x_2 \ .. \ 0]$
$R = \cup_{m \in \{1,2\}} (bnd(c, x_2)(D_m) \cap [\inf_D x_2 \ .. \ \sup_D x_2])$

- if $c \equiv x_1 = x_2 \times x_2 \wedge x_2 \geq 0$, the propagators are defined as for $c \equiv x_1 = x_2 \times x_2$.

- if $c \equiv \texttt{alldifferent}([x_1, \ldots, x_n])$ possible bounds propagators are given by Puget [15] and Mehlhorn and Thiel [13].

Figure 1: Definition of bounds-consistency propagators.

PROPOSITION 2.9. *Let F_1 and F_2 be two bounds-equivalent sets of propagators. For any domain D, then $solv(F_1, D)$ is a false domain iff $solv(F_2, D)$ is a false domain.* □

With respect to search, this proposition can be interpreted as follows. Bounds-equivalent sets of propagators lead to the same failed nodes. As long as the labelling only considers bound information and adds inequality constraints (as is the case for `labelling`), the alternative constraints created during labelling are the same for bounds-equivalent sets of propagators. Taking both facts together, it becomes clear that bounds-equivalent sets of propagators lead to the same search space.

We are now in a position to examine the domain and bounds propagators for individual primitive constraints and determine relationships between them. The first lemma is obvious, and more or less part of the folklore.

LEMMA 2.10. *Let $c \equiv \Sigma_{i=1}^{n} a_i x_i \leq d$, and $x = x_j$ for some $1 \leq j \leq n$. Then $\{dom(c, x)\}$ and $\{bnd(c, x)\}$ are equivalent.* □

Unfortunately there is little further we can go with just the concepts introduced. While it is clear that we can extend the above theorem to any conjunction of inequalities since they are equivalent, this is not possible with bounds-equivalent sets of propagators.

In order to proceed we need to understand what propagators will give the same behaviour when applied to two bounds-equivalent domains. We introduce endpoint-relevance which captures the idea of a set of propagators in whose result the endpoints of the domain support each other, hence the parts of the domain except the endpoints are not relevant to the propagators bounds behaviour.

DEFINITION 2.11. *A set of propagators F is* endpoint-relevant *if for all domains D, if $D_1 = solv(F, D)$ and $D_2 \stackrel{b}{\equiv} D_1$ then $solv(F, D_2) \stackrel{b}{\equiv} D_1$.*

Note that, crucially, endpoint-relevant propagator sets only have special properties at fixpoints of the set of propagators. Otherwise the notion is too strong.

EXAMPLE 2.12. *The set $\{dom(x_1 = 2x_2, x_2), dom(x_1 = 2x_2, x_1)\}$ is endpoint-relevant. Endpoint-relevance requires that endpoints are supported only by other endpoints. Note that $\{dom(x_1 = 2x_2, x_2)\}$ is not endpoint-relevant by itself, consider $D_1(x_1) = [1 .. 7], D_1(x_2) = [1 .. 3]$ and $D_2(x_1) = \{1, 3, 4, 5, 7\}, D_2(x_2) = [1 .. 3]$.* □

Because disequalities have very weak propagators there is a strong correspondence between their domain and bounds propagators.

LEMMA 2.13. *Let $c \equiv \Sigma_{i=1}^{n} a_i x_i \neq d$. Let $x = x_j$ for some $1 \leq j \leq n$. Then $dom(c, x)$ and $bnd(c, x)$ are bounds-equivalent and endpoint-relevant.*

PROOF. Both $f_1 = dom(c, x)$ and $f_2 = bnd(c, x)$ only depend on the endpoints of the input domain since they only remove a value d when each variable in $vars(c) - \{x\}$ has a fixed value (in which case the bounds are equal). Hence they are both endpoint-relevant.

The only difference between f_1 and f_2 is when the non domain-consistent value d is neither a lower nor upper bound. In either case the resulting bounds do not change. □

Two variable equations are also endpoint-relevant because they only involve two variables, and there is a one-to-one correspondence between the values in any solution.

LEMMA 2.14. *Let c be a linear integer equation of the form $b_1 x_1 + b_2 x_2 = e$. Then $\{dom(c, x_1), dom(c, x_2)\}$ and $\{bnd(c, x_1), bnd(c, x_2)\}$ are bounds-equivalent and endpoint-relevant.*

PROOF. Assume w.l.o.g. that $b_1 > 0$. If this is not the case we can replace x_1 by new variable $-x_1'$ and assume $D(x_1') = \{-d \mid d \in D(x_1)\}$. Similarly assume $b_2 > 0$.

Let $D_1 = solv(\{dom(c, x_1), dom(c, x_2)\}, D)$ and $D_2 = solv(\{bnd(c, x_1), bnd(c, x_2)\}, D)$. First by definition $D_1 = iter(\{dom(c, x_1), dom(c, x_2)\}, D)$. Hence

$$d_1 \in D_1(x_1) \quad \text{iff} \quad \frac{e - b_1 d_1}{b_2} \in D(x_2) \tag{1}$$

$$d_2 \in D_1(x_2) \quad \text{iff} \quad \frac{e - b_2 d_2}{b_1} \in D(x_1) \tag{2}$$

Clearly also $b_1 \inf_{D_1} x_1 + b_2 \sup_{D_1} x_2 = e$ and $b_1 \sup_{D_1} x_1 + b_2 \inf_{D_1} x_2 = e$. By the definition of bounds propagation we have that $b_1 \inf_{D_2} x_1 + b_2 \sup_{D_2} x_2 = e$ and $b_1 \sup_{D_2} x_1 + b_2 \inf_{D_2} x_2 = e$. This shows that both sets of propagators are endpoint-relevant.

Now because the endpoints match the conditions of (1) and (2) we have that $\{\inf_{D_2} x_1, \sup_{D_2} x_1\} \subseteq D_1(x_1)$ and similarly for x_2.

Let $D_2^0 = D$, $D_2^{2i+1} = iter(bnd(c, x_1), D_2^{2i})$ and $D_2^{2i+2} = iter(bnd(c, x_2), D_2^{2i+1}), i \geq 0$. We show by induction that $\inf_{D_2^k} x_j \leq \inf_{D_1} x_j$ for $j = 1, 2$ and $k \geq 0$ and $\sup_{D_2^k} x_j \geq \sup_{D_1} x_j$ for $j = 1, 2$ and $k \geq 0$. The base case is straightforward. Suppose $D_2^{k+1}(x_j) \neq D_2^k(x_j)$. We show that the result still holds for D_2^{k+1}. We consider the case when the lower bound of x_1 changes, the other cases are similar. The new lower bound is $\inf_{D_2^{k+1}} x_1 = \lceil \frac{e - b_2 \sup_{D_2^k} x_2}{b_1} \rceil$. Now by induction hypothesis $b_2 \sup_{D_2^k} x_2 \geq b_2 \sup_{D_1} x_2$ and $\inf_{D_1} x_1 = \frac{e - b_2 \sup_{D_1} x_2}{b_1}$ hence $\inf_{D_2^{k+1}} x_1 \leq \inf_{D_1} x_1$.

Finally there exists $k > 0$ s.t. $D_2^k = D_2$ by the definition of D_2. □

Similarly, positive squaring constraints are endpoint-relevant.

LEMMA 2.15. *Let c be of the form $x_1 = x_2 \times x_2 \wedge x_2 \geq 0$. Then $\{dom(c, x_1), dom(c, x_2)\}$ and $\{bnd(c, x_1), bnd(c, x_2)\}$ are bounds-equivalent and endpoint-relevant.*

PROOF. Let $D_1 = solv(\{dom(c, x_1), dom(c, x_2)\}, D)$ and $D_2 = solv(\{bnd(c, x_1), bnd(c, x_2)\}, D)$. First by definition $D_1 = iter(\{dom(c, x_1), dom(c, x_2)\}, D)$. Hence

$$d_1 \in D_1(x_1) \quad \text{iff} \quad \sqrt{d_1} \in D(x_2) \tag{3}$$

$$d_2 \in D_1(x_2) \quad \text{iff} \quad d_2 \times d_2 \in D(x_1) \tag{4}$$

Clearly also $\inf_{D_1} x_1 = \inf_{D_1} x_2 \times \inf_{D_1} x_2$ and $\sup_{D_1} x_1 = \sup_{D_1} x_2 \times \sup_{D_1} x_2$. By the definition of bounds propagation we have that $\inf_{D_2} x_1 = \inf_{D_2} x_2 \times \inf_{D_2} x_2$ and $\sup_{D_2} x_1 = \sup_{D_2} x_2 \times \sup_{D_2} x_2$. This show that both sets of propagators are endpoint-relevant.

Now because the endpoints match the conditions of (3) and (4) we have that $\{\inf_{D_2} x_1, \sup_{D_2} x_1\} \subseteq D_1(x)$ and similarly for x_2.

Let $D_2^0 = D$, $D_2^{2i+1} = iter(bnd(c, x_1), D_2^{2i})$ and $D_2^{2i+2} = iter(bnd(c, x_2), D_2^{2i+1}), i \geq 0$. We show by induction that $\inf_{D_2^k} x_j \leq \inf_{D_1} x_j$ for $j = 1, 2, k \geq 0$ and $\sup_{D_2^k} x_j \geq \sup_{D_1} x_j$ for $j = 1, 2, k \geq 0$. The base case is straightforward. Suppose $D_2^{k+1}(x_j) \neq D_2^k(x_j)$. We show that the result still holds for D_2^{k+1}. We consider the case when the lower bound of x_1 changes, the other cases are similar. The new lower bound is $\inf_{D_2^{k+1}} x_1 = \lceil \inf_{D_k^2} x_2 \times \inf_{D_k^2} x_2 \rceil$. Now by induction $\inf_{D_2^k} x_2 \leq \inf_{D_1} x_2$, hence $\inf_{D_2^{k+1}} x_1 \leq \inf_{D_1} x_1$.

Finally there exists $k > 0$ s.t. $D_2^k = D_2$ by the definition of D_2. □

The above results for endpoint-relevance and bounds-equivalence extend straightforwardly to any two variable primitive constraint describing a continuous bijection (over its co-domain), for example $x_1 = x_2 \times x_2 \times x_2$, $x_1 = a \times x_2 \times x_2 \wedge x_2 \geq 0$, and $x_1 = -x_2^4 - x_2^3 - x_2^2 - x_2 - 1 \wedge x_2 \geq 1$. Three variable constraints are in general not endpoint-relevant.

EXAMPLE 2.16. *The linear equation $x_1 = 3x_2 + 5x_3$ from Example 2.2 is not endpoint-relevant. The solutions $\theta_1 = \{x_1 \mapsto 3, x_2 \mapsto 1, x_3 \mapsto 0\}$ and $\theta_2 = \{x_1 \mapsto 5, x_2 \mapsto 0, x_3 \mapsto 1\}$ illustrate the non-endpoint relevance.*

Note that even $x_1 + x_2 = x_3$ is not endpoint-relevant. Consider the domain-consistent domain $D(x_1) = \{3, 5, 7, 8\}$, $D(x_2) = \{4, 12, 15\}$, $D(x_3) = \{9, 11, 15, 20\}$ and the bounds-equal $D'(x_1) = \{3, 8\}$, $D'(x_2) = \{4, 15\}$, $D'(x_3) = \{9, 20\}$. □

The importance of endpoint-relevance is that we can show that conjoining sets of bounds-equivalent and endpoint-relevant propagators maintains these properties.

THEOREM 2.17. *If F_1 and F_2 are bounds-equivalent and endpoint-relevant and F_1' and F_2' are bounds-equivalent and endpoint-relevant, then $F_1 \cup F_1'$ is bounds-equivalent to $F_2 \cup F_2'$ and both $F_1 \cup F_1'$ and $F_2 \cup F_2'$ are endpoint-relevant.*

PROOF. The proof that $F_j \cup F_j'$ is endpoint-relevant given both F_j and F_j' are endpoint-relevant is straightforward.

We construct a series of bounds-equivalent domains beginning from an arbitrary domain D.

Define $D_1^0 = D$, $D_2^0 = D$ and $D_3^0 = D$. Define $D_j^{2k+1} = solv(F_j, D_3^{2k}), k \geq 0$, for $j = 1, 2$. Define D_4^i to be the range domain such that $D_4^i \stackrel{b}{\equiv} D_1^i$. Define $D_3^i = D_3^i \sqcap D$. Define $D_j^{2k} = solv(F_j', D_3^{2k-1}), k > 0$, for $j = 1, 2$.

We show that $D_1^i \stackrel{b}{\equiv} D_2^i \stackrel{b}{\equiv} D_3^i, i \geq 0$. The base case is trivial.

Now since F_1 and F_2 are bounds-equivalent then $D_1^{2k+1} \stackrel{b}{\equiv} D_2^{2k+1}$ and clearly $D_4^{2k+1} \stackrel{b}{\equiv} D_1^{2k+1}$. By definition $D_1^{2k+1} \sqsubseteq D_3^{2k} \sqsubseteq D$ hence the endpoints of D_1^{2k+1} are in D. Thus $D_3^{2k+1} \stackrel{b}{\equiv} D_4^{2k+1} \stackrel{b}{\equiv} D_1^{2k+1}$.

Similarly since F_1' and F_2' are bounds-equivalent the result holds for $i = 2k, k > 0$.

We must finally reach an i such that $D_3^{i+1} = D_3^i$. Let $D^* = D_3^i$. Clearly then $solv(F_1 \cup F_1', D^*) \stackrel{b}{\equiv} D^*$ since both F_1 and F_1' are endpoint-relevant. Similarly $solv(F_2 \cup F_2', D^*) \stackrel{b}{\equiv} D^*$.

It remains to show that $solv(F_j \cup F_j', D) \stackrel{b}{\equiv} D^*, j = 1, 2$. Clearly since $D^* \sqsubseteq D$ we have that $D^* \stackrel{b}{\equiv} solv(F_j \cup F_j', D^*) \sqsubseteq solv(F_j \cup F_j', D)$ by the monotonicity of $solv$. We now prove that $solv(F_j \cup F_j', D) \sqsubseteq D^*$

We consider the case when $j = 1$, the case when $j = 2$ is identical. We now consider the sequence D_5^i defined as follows: $D_5^0 = D$, $D_5^{2k+1} = solv(F_1, D_5^{2k}), k \geq 0$, and $D_5^{2k} = solv(F_1', D_5^{2k-1}), k > 0$. We show that $D_5^i \sqsubseteq D_3^i, i \geq 0$. The base case is obvious. Clearly $D_5^{2k+1} = solv(F_1, D_5^{2k}) \sqsubseteq solv(F_1, D_3^{2k}) = D_1^{2k+1}$ since $solv$ is monotonic. Now by definition $D_1^{2k+1} \sqsubseteq D_3^{2k+1}$, hence the induction hypothesis holds. The same reasoning applies to D_5^{2k} and D_3^{2k} for $k > 0$. Now there exists i s.t. $D_5^i = solv(F_j \cup F_j', D) \sqsubseteq D_3^i = D^*$. □

EXAMPLE 2.18. *We can now prove that domain propagation or bounds propagation on the example in the introduction*

$$[x_1, x_2, x_3, x_4] :: [0..10], x_1 \leq x_2, 2x_2 = 3x_3 + 1, x_3 \leq x_4$$

is bounds-equivalent. The domain and bounds propagators for $2x_2 = 3x_3 + 1$ are bounds-equivalent and endpoint-relevant by Lemma 2.14, and each of the propagators for $x_1 \leq x_2$ and $x_3 \leq x_4$ are endpoint-relevant and equivalent (domain versus bounds). Hence the conjunction is also bounds-equivalent and endpoint-relevant by Theorem 2.17. □

Typically domain propagation is not ever used for linear equations with more than two variables. This results from the fact that finding the solutions to a linear integer equation is NP-hard. Under the assumption that all linear equations involving more than 2 variables are handled using bounds propagation, we already have enough to show a somewhat surprising result. Using domain propagation (modulo the above discussion) or bounds propagation on linear integer constraints is bounds-equivalent.

PROPOSITION 2.19. *Let C be a conjunction of linear integer constraints excluding linear equations with 3 or more variables. Let C' be a conjunction of linear equations with 3 or more variables.*

Then $\{dom(c, x) \mid c \in C, x \in vars(c)\} \cup \{bnd(c, x) \mid c \in C', x \in vars(c)\}$ is bounds-equivalent to $\{bnd(c, x) \mid c \in C \cup C', x \in vars(c)\}$. □

EXAMPLE 2.20. *How can we go beyond endpoint-relevance. Consider this variation of the example from the introduction*

$$[x_1, x_2, x_3, x_4] :: [0..10], x_1 \leq x_2, x_2 + x_3 = x_4, x_3 \leq x_4$$

The domain and bounds propagators for the constraint $x_2 + x_3 = x_4$ are not endpoint-relevant nor bounds-equivalent. Yet clearly the only constraint that can generate holes in the domains is $x_2 + x_3 = x_4$. But these holes in the domains are irrelevant to the other constraints. Hence domain propagation or bounds propagation for this constraint should be equivalent.

Similarly if we added the constraint $x_1 \neq 3$, then although it generates a hole in the domain of x_1, this is irrelevant to the constraint $x_2 + x_3 = x_4$. Again we should be able to use bounds propagation rather than domain propagation. □

Hence we introduce the notion of range-equivalence, which ensures that the sets of propagators give bounds-equivalent answers for range domains.

DEFINITION 2.21. *Two sets of propagators F_1 and F_2 are* range-equivalent*, iff for each range domain D, $solv(F_1, D) \stackrel{b}{\equiv} solv(F_2, D)$.*

Clearly range-equivalent propagators detect failure at the same time for input range domains.

PROPOSITION 2.22. *Let F_1 and F_2 be two range-equivalent sets of propagators. For any range domain D, $solv(F_1, D)$ is a false domain, iff $solv(F_2, D)$ is a false domain.* □

We will be interested in determining which sets of propagators are range-equivalent.

LEMMA 2.23. *If c is a linear equation $\sum_{i=1}^{n} a_i x_i = d$ with $|a_i| = 1, 1 \leq i \leq n$, then $\{dom(c, x_j)\}$ and $\{bnd(c, x_j)\}$ are bounds-preserving and range-equivalent.*

PROOF. Assume w.l.o.g. that $a_j = 1$. Let $[l \,..\, u] = bnd(c, x_j)(D)$. By definition

$$\begin{aligned} l &= d - \sum_{i=1, i\neq j}^{n} \sup_D(a_i x_i) \\ u &= d - \sum_{i=1, i\neq j}^{n} \inf_D(a_i x_i) \end{aligned}$$

We show that for each $d_j \in [l \,..\, u]$ there is a solution $\theta \in D$ of $\sum_{i=1, i\neq j}^{n} a_i x_i = d - d_j$. This proves that $dom(c, x_j)$ is bounds-preserving, and that $dom(c, x_j)(D) = bnd(c, x_j)(D)$

Clearly $\theta_1 = \{x_i \mapsto \sup_D(a_i x_i)\}$ is a solution when $d_j = l$, and $\theta_1 = \{x_i \mapsto \inf_D(a_i x_i)\}$ is a solution when $d_j = u$ by their definition. Now

$$\begin{aligned} u - l &= \theta_2(d - \sum_{i=1, i\neq j}^{n} a_i x_i) - \theta_1(d - \sum_{i=1, i\neq j}^{n} a_i x_i - d) \\ &= \sum_{i=1, i\neq j}^{n} (\sup_D(a_i x_i) - \inf_D(a_i x_i)) \end{aligned}$$

Take $l < d_j < u$ then $u - d_j < u - l$ and hence there exist $e_i \leq \sup_D(a_i x_i) - \inf_D(a_i x_i)$ such that $u - d_j = \sum_{i=1, i\neq j}^{n} e_i$ Now $e_i + \inf_D(a_i x_i) \in D(x_i)$, since $D(x_i)$ is a range and $\theta = \{x_i \mapsto e_i + \inf_D(a_i x_i)\}$ is a solution of $\sum_{i=1, i\neq j}^{n} a_i x_i = d - d_j$ by construction. □

Clearly Examples 2.2 and 2.3 illustrate that the domain and bounds propagators for linear integer equations with arbitrary coefficients are not range-equivalent. Of more interest is the fact that we can produce efficient range-equivalent propagators for complex constraints like `alldifferent`.

LEMMA 2.24. *If $c \equiv$ `alldifferent`$([x_1, \ldots, x_n])$, then the sets $\{dom(c, x_i) \mid 1 \leq i \leq n\}$ and $\{bnd(c, x_i) \mid 1 \leq i \leq n\}$ are range-equivalent.*

PROOF. See for example Puget [15]. □

Once we have established range-equivalence for primitive constraints we can extend this to larger sets of constraints using Theorem 2.26 below. There are some side conditions about the interaction of variables that first require definition.

DEFINITION 2.25. *A variable x is* bounds-only *w.r.t a set of propagators F, if each propagator $f \in F$ for variable x is bounds-only.*

A set of propagators F is endpoint-relevant *for variables V, if for all domains D, if $D_1 = solv(F, D)$ and $D_2 \stackrel{b}{\equiv} D_1$ and $D_2 =_{\mathcal{V}-V} D_1$, then $solv(F, D_2) \stackrel{b}{\equiv} D_1$.*

THEOREM 2.26. *Let F_1 and F_2 be range-equivalent. Let F_1' and F_2' be range-equivalent. Let F_1' and F_2' be endpoint-relevant and bounds-only on variables $V = vars(F_1) \cup vars(F_2)$. Then $F_1 \cup F_1'$ and $F_2 \cup F_2'$ are range-equivalent.*

PROOF. We construct a series of bounds-equivalent domains beginning from an arbitrary range domain D.

Define $D_1^0 = D$, $D_2^0 = D$ and $D_3^0 = D$. Define $D_j^{2k+1} = solv(F_j, D_3^{2k}), k \geq 0$, for $j = 1, 2$. Define D_3^i to be the range domain such that $D_3^i \stackrel{b}{\equiv} D_1^i$. Define $D_j^{2k} = solv(F_j', D_3^{2k-1})$, $k > 0$, for $j = 1, 2$.

We show that $D_1^i \stackrel{b}{\equiv} D_2^i \stackrel{b}{\equiv} D_3^i, i \geq 0$. The base case is trivial.

Now since F_1 and F_2 are range-equivalent, $D_1^{2k+1} \stackrel{b}{\equiv} D_2^{2k+1}$ and clearly $D_3^{2k+1} \stackrel{b}{\equiv} D_1^{2k+1}$.

Similarly since F_1' and F_2' are range-equivalent the result holds for $i = 2k, k > 0$.

We must finally reach an i such that $D_3^{i+1} = D_3^i$. Let $D^* = D_3^i$. Let $E_j = solv(F_j', D^*)$. Then $E_j =_V D^*$ since F_j' is bounds-only on V and by the definition of D^*. Also $E_1 \stackrel{b}{\equiv} E_2$ since F_1' and F_2' are range-equivalent. Let $E_j' = solv(F_j, E_j)$. Clearly $E_1' \stackrel{b}{\equiv} E_2'$ since $E_1 =_V D^* =_V E_2$ and both F_1 and F_2 only depend on variables in V. And by the definition of D^*, $E_j' =_V D^*$.

In fact $E_j' = solv(F_j \cup F_j', D)$. Since F_j' is endpoint-relevant and $E_j' =_V D^*$ we have that $solv(F_j', E_j') = E_j'$. Clearly E_j' is a fixpoint of $\lambda x.solv(F_j \cup F_j', x \sqcap D)$, for $j = 1, 2$ and hence $E_j' \sqsubseteq solv(F_j \cup F_j', D)$. It remains to show that $solv(F_j \cup F_j', D) \sqsubseteq E_j'$.

We consider the case when $j = 1$, the case when $j = 2$ is identical. We now consider the sequence D_5^i defined as follows: $D_5^0 = D$, $D_5^{2k+1} = solv(F_1, D_5^{2k}), k \geq 0$, and $D_5^{2k} = solv(F_1', D_5^{2k-1}), k > 0$. We show that $D_5^i \sqsubseteq D_3^i, i \geq 0$. The base case is obvious. Clearly $D_5^{2k+1} = solv(F_1, D_5^{2k}) \sqsubseteq solv(F_1, D_3^{2k}) = D_1^{2k+1}$ since $solv$ is monotonic. Now by definition $D_1^{2k+1} \sqsubseteq D_3^{2k+1}$, hence the induction hypothesis holds. The same reasoning applies to D_5^{2k} and D_3^{2k} for $k > 0$. Now there exists i s.t. $D_5^i = solv(F_j \cup F_j', D) \sqsubseteq D_3^i = D^*$. The final two steps follow from the definition of E_1'. □

EXAMPLE 2.27. *Consider two range-equivalent sets of propagators for the constraint* `alldifferent`$([x_1, x_2, x_3])$*, one set, F_1, based on domain propagation and the other, F_2, based on bounds propagation.*

Let F_1' be domain propagators for $x_1 \leq x_3, 2x_3 + x_4 \leq 6, x_2 + x_5 \leq 4, x_4 = 2x_5 - 1$, and F_2' be domain propagators for the same system. By Lemma 2.14 and Theorem 2.17 we have that F_1' and F_2' are bounds-equivalent and endpoint-relevant. Clearly on the variables x_1, x_2 and x_3 they are bounds-only.

Consider the initial domain $D(x_i) = [1 \,..\, 3], 1 \leq i \leq 5$. Domain propagation of F_1' obtains $D_1^1(x_1) = [1 \,..\, 2]$, $D_1^1(x_2) = [1 \,..\, 3]$, $D_1^1(x_3) = [1 \,..\, 2]$, $D_1^1(x_4) = \{1, 3\}$, and $D_1^1(x_5) = [1 \,..\, 2]$. Bounds propagation of F_2' obtains the

range domain D_2^1 that is bounds-equivalent to D_1^1. Note that $D_1^1 =_{\{x_1,x_2,x_3\}} D_2^1$.

Domain propagation on F_1 then determines domain D_1^2 which modifies $D_1^2(x_2) = \{3\}$. Similarly for F_2 applies to D_2^1 obtaining D_2^2.

Domain propagation of F_1' now obtains D_1^3 which modifies $D_1^3(x_4) = \{1\}$ and $D_1^3(x_5) = \{1\}$. Similarly for F_2 applies to D_2^2 obtaining D_2^3. Now both fixpoints are reached and $D_2^3 = D_1^3$. □

EXAMPLE 2.28. *Consider the following well-known program for $SEND + MORE = MONEY$.*

```
smm(S,E,N,D,M,O,R,Y) :-
     [S,E,N,D,M,O,R,Y] ::  [0..9],
     S >= 1, M >= 1,
                  1000 * S + 100 * E + 10 * N + D
                + 1000 * M + 100 * O + 10 * R + E
     = 10000 * M + 1000 * O + 100 * N + 10 * E + Y,
     alldifferent([S,E,N,D,M,O,R,Y]),
     labelling([S,E,N,D,M,O,R,Y]).
```

Assuming that bounds propagation is used for the linear equations with more than 2 variables, then using either bounds or domain propagation for `alldifferent` *leads to the same search space being traversed. The result holds using Lemma 2.24 and Theorem 2.26.* □

3. ANALYSING FD PROGRAMS

Now we are ready to devise a bottom-up analysis to discover weaker sets of propagators for a CLP(FD) program that give equivalent search behaviour. We assume we are given a pure CLP(FD) program and must choose for each primitive constraint which set of propagators to use.

For simplicity, we only consider pure CLP(FD) programs without data structures. We could extend the approach to CLP(FD) programs with types defined by deterministic finite tree automata using the methodology of [9].

Analysis and "optimization" of the CLP(FD) program proceeds in two phases.

Range and Endpoint In the first phase, a bottom-up analysis determines which variables are guaranteed to have a range domain, and which are guaranteed to only appear in endpoint-relevant constraints.

Calling context In the next phase, we determine the calling context (in terms of range and endpoint information) for each literal, and replace it by the appropriate propagation method.

We assume the reader is somewhat familiar with abstract interpretation of CLP programs (see e.g. [5, 10]).

3.1 Range and Endpoint Descriptions

The first phase is a simple bottom-up abstract interpretation where we determine which variables must have range domains, and which variables are only involved in endpoint-relevant constraints.

The bottom-up analysis determines for each user-defined constraint $p(x_1, \ldots, x_n)$ two Boolean formulae[1] describing its (non-)range and (non-)endpoint behaviour. The intuition is that the Boolean variable corresponding to a variable x is *true* if the variable is not guaranteed to have a range domain (resp. not guaranteed to appear in only endpoint-relevant constraints).

[1]For most of the primitive constraints we could simply restrict ourselves to conjunctions of positive literals (i.e. sets of Boolean variables). We use the Boolean domain for generality.

Table 1: Range and endpoint descriptions for primitive constraints.

Constraint	α_R	α_E
true	*true*	*true*
$\sum_{i=1}^n a_i x_i \leq d$	*true*	*true*
$x_1 = d$	*true*	*true*
$a_1x_1 + a_2x_2 = d$, $\lvert a_i\rvert = 1$	$x_1 \leftrightarrow x_2$	*true*
$a_1x_1 + a_2x_2 = d$	$x_1 \wedge x_2$	*true*
$\sum_{i=1}^n a_i x_i = d, n > 2^2$	$\wedge_{i=1}^n x_i$	$\wedge_{i=1}^n x_i$
$\sum_{i=1}^n a_i x_i = d$, $n > 2$, $\lvert a_i\rvert = 1$	$\wedge_{i=2}^n (x_1 \leftrightarrow x_i)$	$\wedge_{i=1}^n x_i$
$\sum_{i=1}^n a_i x_i \neq d$	$\wedge_{i=1}^n x_i$	*true*
$x_1 = x_2 \times x_3$	$\wedge_{i=1}^3 x_i$	$\wedge_{i=1}^3 x_i$
$x_1 = x_2 \times x_2 \wedge x_2 \geq 0$	$x_1 \wedge x_2$	*true*
$x_1 = x_2 \times x_2$	$x_1 \wedge x_2$	$x_1 \wedge x_2$
`alldifferent`$([x_1, \ldots, x_n])$	$\wedge_{i=1}^n x_i$	$\wedge_{i=1}^n x_i$
`default`$([x_1, \ldots, x_n])$	$\wedge_{i=1}^n x_i$	$\wedge_{i=1}^n x_i$
`labelling`$([x_1, \ldots, x_n])$	*true*	*true*
`labellingff`$([x_1, \ldots, x_n])$	*true*	$\wedge_{i=1}^n x_i$
`labellingmid`$([x_1, \ldots, x_n])$	$\wedge_{i=1}^n x_i$	$\wedge_{i=1}^n x_i$

EXAMPLE 3.1. *The (non-)range description of $x_1 \leq x_2$ is true since each of the variables appearing in it is guaranteed to have a range domain. The (non-)range description of $x_4 = 2x_5 - 1$ is $x_4 \wedge x_5$ indicating that both x_4 and x_5 may not have range domains. Its (non-)endpoint description is true indicating that x_4 and x_5 only appear in endpoint-relevant constraints.* □

DEFINITION 3.2. *The abstract domain A for both descriptions used is a simple (inverted) domain of Boolean formulae defined as follows:*

$$\phi_1 \sqsubseteq_A \phi_2 \text{ iff } \phi_2 \rightarrow \phi_1. \quad \bot_A = true,\ \top_A = false.$$
$$\phi_1 \sqcap_A \phi_2 = \phi_1 \vee \phi_2. \quad \phi_1 \sqcup_A \phi_2 = \phi_1 \wedge \phi_2.$$

DEFINITION 3.3. *The meaning of a range description ϕ is defined by the concretization function γ_R:*

$$\gamma_R(\phi) = \{C \mid \forall x \text{ s.t. } (\forall x\phi) \leftrightarrow (\exists x\phi), \forall c \in C, dom(c,x) \text{ is bounds-only}\}$$

The condition $(\forall x\phi) \leftrightarrow (\exists x\phi)$ holds whenever ϕ does not constrain x in any way.

The meaning of a Endpoint description ϕ is defined by the concretization function γ_E:

$$\gamma_E(\psi) = \{\, C \mid \forall x \text{ s.t. } (\forall x\phi) \leftrightarrow (\exists x\phi), \forall c \in C, x \in vars(c), \{dom(c,y) \mid y \in vars(c)\} \text{ is endpoint-relevant}\}$$

We can define the approximation function α for the range and endpoint descriptions for each primitive constraint as in Table 1. Here we treat `labelling` goals, which drive the search for solution, as primitive constraints since their implementation involves data-structure manipulation.

[2]We often *apriori* choose bounds propagation for these constraints in which case the description is $(true, true)$.

Note that the only interesting Boolean formulae arise from range descriptions for linear equations with unit coefficients, since these constraints are bounds-preserving.

We can lift the analysis to conjunctions of constraints simply by conjoining the descriptions, so abstract conjunction is defined as $\mathsf{Aconj}_R = \mathsf{Aconj}_E = \wedge$. We can similarly (inaccurately) handle disjunctions by conjunction so abstract disjunction is defined as $\mathsf{Adisj}_R = \mathsf{Adisj}_E = \wedge$. Projection of descriptions onto a set of variables V is Boolean existential quantification, $\mathsf{Aproj}_R(V,\phi) = \mathsf{Aproj}_E(V,\phi) = \exists(\mathcal{V} - V)\phi$.

For recursive programs we can find the least fixpoint in the usual manner (see e.g. [10]). Note that since there are no infinite ascending chains this process is finite. We can alternatively (and this is the approach taken in the implementation) use a constraint based fixpoint rule (as in Hindley-Milner type inference, see e.g. [3]) which simply ensures that recursive calls have the same descriptions as the head. This is more inaccurate but sound. We assume a function $\mathsf{analyse}_R(G)$ for analysing a goal G using abstract domain A (one of R or E). Hence $\mathsf{analyse}_R(G)$ and $\mathsf{analyse}_E(G)$ return the range and endpoint descriptions for a goal G.

EXAMPLE 3.4. *Consider the following program:*

```
g(x1,x2,x3,x4,x5) :- x5 ≠ 6, p(x1,x2,x3,x4,x5).
g(x1,x2,x3,x4,x5) :- x4 ≠ 3.
p(x1,x2,x3,x4,x5) :- alldifferent([x1,x2,x3]),
                     q(x1,x2,x3,x4,x5).
q(x1,x2,x3,x4,x5) :- x1 ≤ x6, x6 ≤ x2, 2x3 + x4 ≤ 6,
                     x2 + x5 ≤ 4, x4 = 2x5 − 1.
```

The (range,endpoint) answer descriptions for each literal of the program is shown in the table below:

$x_1 \leq x_6$	$(true, true)$
$x_6 \leq x_2$	$(true, true)$
$2x_3 + x_4 \leq 6$	$(true, true)$
$x_2 + x_5 \leq 4$	$(true, true)$
$x_4 = 2x_5 - 1$	$(x_4 \wedge x_5, true)$
$x_4 \neq 3$	$(x_4, true)$
$x_5 \neq 6$	$(x_5, true)$
$\mathtt{q}(x_1,x_2,x_3,x_4,x_5)$	$(x_4 \wedge x_5, true)$
$\mathtt{alldifferent}([x_1,x_2,x_3])$	$(x_1 \wedge x_2 \wedge x_3, x_1 \wedge x_2 \wedge x_3)$
$\mathtt{p}(x_1,x_2,x_3,x_4,x_5)$	$(x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5, x_1 \wedge x_2 \wedge x_3)$
$\mathtt{g}(x_1,x_2,x_3,x_4,x_5)$	$(x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5, x_1 \wedge x_2 \wedge x_3)$

□

3.2 Determining Calling Contexts

Unlike many analysis-based optimizations, here we need to understand for each primitive constraint, the effect of the remainder of the program on the variables that it involves. This is crucially important in determining whether domain propagation for the constraint will be different to bounds propagation. Even if a constraint can cause holes in the domain of its variables this may be unimportant, if there are no other constraints involving these variables that act differently if holes are present.

Given a primitive constraint and a description of the Range and Endpoint information from the other constraints upon its variables, we can determine when it is safe to use the bounds propagators for the constraint. Table 2 gives the weakest allowable description for each component that allows the bounds propagators to be used.[3] Each of these optimizations is justified by a lemma. The exception is `labellingff`, which we can replace by a version which uses the calculation $\sup_D x - \inf_D x$ rather than $|D(x)|$ to determine the variable with the smallest domain, if all the variables involved are guaranteed to have range domains.

Table 2: Weakest possible descriptions to allow the use of bounds propagators.

Constraint	ϕ_R	ϕ_E	Lemma
$\sum_{i=1}^n a_i x_i \leq d$	*false*	*false*	2.10
$\sum_{i=1}^n a_i x_i \neq d$	*false*	*true*	2.13
$a_1x_1 + a_2x_2 = d$	*false*	*true*	2.14
$\sum_{i=1}^n a_i x_i = d$, $n > 2$, $\lvert a_i\rvert = 1$	*true*	*true*	2.23
$x_1 = x_2 \times x_2 \wedge x_2 \geq 0$	*false*	*true*	2.15
$\mathtt{alldifferent}([x_1,\ldots,x_n])$	*true*	*true*	2.24
$\mathtt{labellingff}([x_1,\ldots,x_n])$	*true*	*true*	—

The calling contexts for each literal in the program are determined using a top-down analysis starting from an initial entry point, say `main`. We can mimic multiple entry points G_1 to G_n by simply defining `main` as

$$main \texttt{ :- } G_1. \quad \ldots \quad main \texttt{ :- } G_n.$$

The analysis starts from the calling pattern `main` : (*true*,*true*).

Given we are processing a calling pattern $p(x_1,\ldots,x_n)$: (CP_R, CP_E), we process each rule of the form

$$p(x_1,\ldots,x_n) \texttt{ :- } A_1,\ldots,A_m$$

by determining the calling context for each literal A_i as the conjunction of the analysis answers for $A_j, 1 \leq i \neq j \leq m$ with the calling description CP. The algorithm is formalized in Figure 2. Initially it is called with an empty table of previous optimizations ($Table$).

EXAMPLE 3.5. *Returning to the program of Example 3.4 and assuming an entry point* $\mathtt{g}(x_1,x_2,x_3,x_4,x_5)$*, transformation determines calling contexts (ignoring inequalities):*

$\mathtt{g}(x_1,x_2,x_3,x_4,x_5)$	:	$(true, true)$
$\mathtt{p}(x_1,x_2,x_3,x_4,x_5)$	:	$(x_5, true)$
$\mathtt{alldifferent}([x_1,x_2,x_3])$	:	$(true, true)$
$\mathtt{q}(x_1,x_2,x_3,x_4,x_5)$	:	$(x_1 \wedge x_2 \wedge x_3 \wedge x_5, x_1 \wedge x_2 \wedge x_3)$
$x_5 \neq 6$	:	$(x_5, true)$
$x_4 \neq 3$	:	$(true, true)$
$x_4 = 2x_5 - 1$	:	$(x_5, true)$

Hence we replace the `alldifferent` *constraint and the constraints* $x_4 \neq 3$ *and* $x_4 = 2x_5 - 1$ *by their bounds propagation versions. The program output by the transformation is*

```
g(x1,x2,x3,x4,x5) :- bnd(x5 ≠ 6), p(x1,x2,x3,x4,x5).
g(x1,x2,x3,x4,x5) :- bnd(x4 ≠ 3).
p(x1,x2,x3,x4,x5) :- bnd(alldifferent([x1,x2,x3])),
                     q(x1,x2,x3,x4,x5).
q(x1,x2,x3,x4,x5) :- bnd(x1 ≤ x6), bnd(x6 ≤ x2),
                     bnd(2x3 + x4 ≤ 6),
                     bnd(x2 + x5 ≤ 4),
                     bnd(x4 = 2x5 − 1).
```

□

[3] So *false* allows any description, while *true* requires that the description is exactly *true*.

```
transform(L : (CP_R, CP_E), Table)
  case L of
  primitive constraint or labelling literal c:
    if (CP_R, CP_E) satisfies conditions in Table 2
      return bnd(c)
    else return dom(c)
  atom p(y_1, ..., y_n):
    if ∃L' : (CP'_R, CP'_E) ↦ L'' ∈ Table and renaming ρ such that ρ(L' : (CP'_R, CP'_E)) = L : (CP_R, CP_E)
      return ρ(L'')
    let p' be a new predicate symbol not in Table
    Table := Table ∪ {p(y_1, ..., y_n) : (CP_R, CP_E) ↦ p'(y_1, ..., y_n)}
    foreach rule p(x_1, ..., x_n) :- A_1, ..., A_m
      let ρ be the renaming {x_i ↦ y_i}
      G := true
      for i = m..1
        CP'_R := Aproj_R(vars(A_i), ρ(CP_R) ∧ ⋀{analyse_R(A_j) | 1 ≤ j ≠ i ≤ m})
        CP'_E := Aproj_E(vars(A_i), ρ(CP_E) ∧ ⋀{analyse_E(A_j) | 1 ≤ j ≠ i ≤ m})
        O_i := transform(A_i : (CP'_R, CP'_E), Table)
        G := (O_i, G)
      output p'(x_1, ..., x_n) :- G
    return p'(y_1, ..., y_n)
```

Figure 2: Algorithm for transforming calling pattern $L : (CP_R, CP_E)$ given previous optimizations in $Table$.

Note that the optimization is multi-variant, that is it can produce multiple specialized versions of the same predicate. The result of the transformation is a new program with the same search space.

THEOREM 3.6. *If P is a CLP(FD) program and P' is the program output for* transform*$(L : (true, true), \emptyset)$ then the search space explored executing goal L using P and P' is the same.*

PROOF. (Sketch) The replacements made in P' are individually justified by the Lemmas shown in Table 2 and Theorems 2.17 and 2.26. This ensures that during execution of the programs each conjunction of propagators collected on an execution path is range-equivalent. Using Proposition 2.22, the propagators on identical paths detect failure at the same time. □

One has to be quite careful to go beyond the transformations allowed here, because interaction of propagators can be subtle.

EXAMPLE 3.7. *Consider the goal*

$$\texttt{alldifferent}([x_1, x_2, x_3, x_4, x_5, x_6]),$$
$$x_6 = x_1 + 3, x_4 = x_1 + 3.$$

Since the equations are bounds-preserving we might assume that the `alldifferent` *bounds and domain propagators will be equivalent. This is not the case. Consider the domain $D(x_1) = D(x_3) = [1 .. 3]$, $D(x_2) = \{2\}$, $D(x_4) = D(x_5) = D(x_6) = [4 .. 6]$ then domain propagation and bounds propagation are not the same. E.g.* `alldifferent` *domain propagator gives $D(x_1) = D(x_3) = \{1, 3\}$, $D(x_2) = \{2\}$, $D(x_4) = D(x_5) = D(x_6) = [4 .. 6]$ but then domain propagation on the equalities gives $D(x_4) = D(x_6) = \{4, 6\}$, Subsequently, the* `alldifferent` *domain propagator gives $D(x_5) = \{5\}$. The* `alldifferent` *bounds propagator gives $D(x_1) = D(x_3) = [1 .. 3]$, $D(x_2) = \{2\}$, $D(x_4) = D(x_5) = D(x_6) = [4 .. 6]$ and there is no further propagation. The results are not bounds-equal.* □

There are a number of obvious ways to improve this analysis. We can eliminate (non-)range information about variables with initial domain of the form $[l .. l + 1]$ (most notably Boolean variables $[0 .. 1]$) since they always have range domains. We can use a preliminary groundness analysis to determine which variables will always be fixed, and then use this information to treat constraints in simpler forms, e.g. the constraint $x_1 = x_2 \times x_3$ becomes a two variable equation, if x_2 is always fixed by the time the constraint is reached.

3.3 Experimental Evaluation

We have constructed a prototype analyser and transformer for pure CLP(FD) programs. Here we give preliminary experiments to illustrate the effect of the transformation.

We illustrate the effect of the transformation on three classes of benchmarks. The first class include NP-hard graph problems and multi-knapsack problems with unit values. The graph examples (for example, see [6]) are vertex cover (`vc-*`) and independent sets (`is-*`) modeled in the natural way using Boolean variables indicating which vertices are in the selected set. The graphs used are random graphs of 20 and 40 nodes. The constraints are all inequalities except the objective function which is defined using a large linear equation with unit coefficients. The multi-knapsack problems (`mk-*`) are similar and use integer variables for the number of selected items. The multiple resource restrictions are expressed by linear inequality constraints, the objective function is again a linear equality involving all variables with unit coefficients. Both instances are based on the data set given in [18, Section 2.1.8]. Analysis shows that we can replace domain propagation on the linear equation with bounds propagation without affecting search space.

The second class includes well-known examples `smm` (see Example 2.28), `donald` ($DONALD + GERALD = ROBERT$), magic squares `magic-5` and Golomb ruler problems (`golomb-*`). Here we assume bounds propagation is used on linear equations of more than three variables. Anal-

Table 3: Comparison of executions for original and transformed versions of the programs

	Original				Transformed				
Program	Nodes	DomChg	Exec	Time	Nodes	DomChg	Exec	Time	Search
`vc-20`	126	767	490	25.6	=	=	=	−73%	best
`vc-40`	6 340	83 292	24 616	4 213.8	=	=	=	−88%	best
`is-20`	94	271	194	21.8	=	=	=	−78%	best
`is-40`	1 542	9 240	3 477	958.8	=	=	=	−89%	best
`mk-1`	135 582	1 650 536	1 584 948	25 988.3	=	=	=	−55%	best
`mk-2`	865 164	13 290 127	9 240 208	133 945.0	=	=	=	−54%	best
`smm`	5	33	26	0.6	=	=	=	−36%	all
`donald`	9 232	22 605	34 910	930.8	=	−13%	+16%	−38%	all
`magic-5`	6 822	95 841	111 208	1 413.8	=	+19%	+21%	−40%	first
`golomb-8`	6 490	254 881	294 816	2 392.1	=	−4%	+3%	−34%	best
`golomb-9`	34 910	1 861 679	2 146 317	18 522.1	=	−7%	−0%	−32%	best
`golomb-10`	191 050	13 721 156	15 747 604	152 272.0	=	−8%	−2%	−32%	best
`sched-bridge`	62	3 942	11 973	22.1	=	+0%	−0%	+36%	best
`sched-orb06`	57 108	2 970 070	6 048 079	59 105.0	=	−5%	−1%	+39%	best
`sched-orb09`	5 648	307 798	623 801	6 436.7	=	−5%	−1%	+36%	best
`sched-la18`	19 174	386 236	898 941	11 167.5	=	−5%	−1%	+34%	best
`sched-mt10`	43 628	3 023 368	6 107 837	49 502.1	=	−7%	−2%	+39%	best

ysis shows we can use bounds propagation on the single `alldifferent` constraint in each benchmark without affecting search space.

The third class of examples are scheduling examples: including the well-known bridge scheduling example [4], the remainder are job-shop scheduling examples taken from J.E. Beasley's OR Library [2]. Here analysis shows that the `cumulativeEF` constraint (using a generalization of edge-finding [12]) appearing in the benchmarks only requires bounds propagation.

All but the multi-knapsack and the scheduling benchmarks use default labelling `labelling`. The labelling for the multi-knapsack problems split the domains of variables according to the arithmetic mean of infimum and supremum of a variable (and thus are very close to the default labelling). The scheduling benchmarks use a labelling strategy similar to that mentioned in [1] (the labelling strategy considers and contributes bounds information only and hence is equivalent to `labelling` for the purposes of the analysis).

Table 3 gives results for executing each benchmark to find either all solutions (all in column Search), the first solution (first), or a best solution (best). The table contains the number of searched nodes, the number of times the domain of a variable was changed (DomChg), the number of times a propagator was executed (Exec), and the runtime (walltime) in milliseconds. The numbers for the transformed programs are given relative to the numbers of the original programs, a negative percentage means that the transformed program shows an improvement of that percentage. For example, a time-value of −50% means that the transformed program is twice as fast. A positive percentage is analogous.

All numbers have been taken with the Mozart (version 1.2.0) implementation of Oz [14] on a standard personal computer with a 700 MHz Athlon processor, 256 MB of main memory, and RedHat Linux 7.1 as operating system. All runtimes are the arithmetic mean of 25 runs.

In order to do the tests we needed to add bounds propagation versions of `alldifferent` and `cumulativeEF` to Mozart. We used the naive $O(n^2)$ algorithm described in [15] for `alldifferent`. We mimicked a range-equivalent version of `cumulativeEF` by using the domain propagation version on a new copy $(x'_1, \ldots, x'_n)$ of the original variables $(x_1, \ldots, x_n)$, and connecting these to the original variables through inequalities $x_i \geq x'_i$ and $x'_i \geq x_i$.

The results show substantial improvement in execution time for the first class of benchmarks illustrating the expense of domain propagation on large equations. The number of nodes explored in each case is identical (illustrating Theorem 3.6 in action) for this and all benchmarks. Moreover the domains in this case are always identical (not just bounds-equal) as illustrated by the number of domain changes and executions.

The results for the second set of benchmarks show a moderate speedup. This is due to the fact that our bounds version of `alldifferent` is only a prototype compared to the mature (already provided) domain version. It can be expected that with a state-of-the-art implementation of a bounds propagation version of `alldifferent` (either the $O(n \log n)$ algorithm of [15], or the linear algorithm of [13]), the improvement in runtime will be considerably better. Note that the number of domain changes reduces for the larger benchmarks (`golomb-*`) indicating useless (in terms of search space) removal of internal values. Interestingly the number of constraint executions can be greater for bounds propagation since it may require a number of executions to determine the same information as the domain version.

For the third class of benchmarks, we obviously expect a slow down since we are mimicking a range-equivalent bounds propagation version of `cumulativeEF` using the domain version. However, bounds propagation requires less variable modifications as well as less constraint executions. This suite shows that it is worth investigating bounds propagators for `cumulativeEF` which are range-equivalent to the current domain version.

4. CONCLUSION

We have examined the propagation behaviour of domain and bounds propagators for common primitive constraints, and discovered cases where they will determine failure at

the same time. By constructing theorems about how conjunctions of propagators can be built which maintain this property we are able to prove when domain and bounds propagation for a constraint system will give the same behaviour. We devised an analysis to determine where we can safely replace domain propagation by bounds propagation in a CLP(FD) program. We have illustrated a number of real programs where the analysis is able to determine weaker forms of propagators with equivalent search behaviour, and gave some evidence for the improvements possible.

There is plenty more scope for understanding when one form of propagator is equivalent in strength to another. We should characterise the many global constraints available like `alldifferent` and `cumulativeEF` in terms of their propagation behaviour, and extend the analysis to handle them. The most important use of this information is probably in building more efficient versions of global constraints and recognizing where they can be used safely without increasing search space. There is also scope for finding weaker conditions that maintain bounds-equality of domains for constraints.

Other kinds of propagation are also worth considering such as value propagation or propagators for stronger notions of consistency like path consistency.

There is further scope for improving the propagators produced by the transformation. For example, consider the constraints

$$x_1 + x_2 + x_3 \leq 3, x_3 + x_4 \neq 2, \texttt{alldifferent}([x_4, x_5, x_6])$$

We can safely use the bounds propagator $bnd(x_3 + x_4 \neq 2, x_3)$ for one variable in the disequality while using the domain propagator $dom(x_3 + x_4 \neq 2, x_4)$ for the other variable.

Acknowledgements

We are grateful to Tobias Müller for support with mimicking bounds-propagation. We also thank the anonymous referees for their suggestions which undoubtedly improved the paper.

5. REFERENCES

[1] P. Baptiste, C. Le Pape, and W. Nuijten. Incorporating efficient operations research algorithms in constraint-based scheduling. In *First International Joint Workshop on Artificial Intelligence and Operations Research*, 1995.

[2] J. E. Beasley. OR library. `http://www.ms.ic.ac.uk/info.html`.

[3] B. Demoen, M. García de la Banda, and P. J. Stuckey. Type constraint solving for parametric and ad-hoc polymorphism. In J. Edwards, editor, *Proceedings of the 22nd Australian Computer Science Conference*, pages 217–228. Springer-Verlag, January 1999.

[4] M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving Large Combinatorial Problems in Logic Programming. *The Journal of Logic Programming*, 8(1-2):74–94, Jan.-Mar. 1990.

[5] M. García de la Banda, M. Hermenegildo, M. Bruynooghe, V. Dumortier, G. Janssens, and W. Simoens. Analysis of constraint logic programs. *ACM Transactions on Programming Languages and Systems*, 18(5):564–614, 1996.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman And Company, New York, NY, USA, 1979.

[7] W. Harvey and P. J. Stuckey. Constraint representation for propagation. In M. Maher and J.-F. Puget, editors, *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming*, volume 1520 of *Lecture Notes in Computer Science*, pages 235–249. Springer-Verlag, Oct. 1998.

[8] A. D. Kelly, K. Marriott, A. Macdonald, P. J. Stuckey, and R. Yap. Optimizing compilation for CLP($\mathcal{R}$). *ACM Transactions on Programming Languages and Systems*, 20(6):1223–1250, 1998.

[9] V. Lagoon and P. J. Stuckey. A framework for analysis of typed logic programs. In *Proceedings of the Fifth International Symposium on Functional and Logic Programming*, volume 2024 of *Lecture Notes in Computer Science*, pages 296–310. Springer-Verlag, 2001.

[10] K. Marriott and H. Søndergaard. Analysis of constraint logic programs. In S. Debray and M. Hermengildo, editors, *Logic Programming: Proceedings of the 1990 North American Conference*, pages 531–547, Austin, TX, USA, October 1990. The MIT Press.

[11] K. Marriott and P. J. Stuckey. *Programming with Constraints: an Introduction*. The MIT Press, 1998.

[12] P. Martin and D. B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization, 5th International IPCO Conference*, volume 1084 of *Lecture Notes in Computer Science*, pages 389–403, Vancouver, BC, Canada, June 1996. Springer-Verlag.

[13] K. Mehlhorn and S. Thiel. Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. In R. Dechter, editor, *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*, volume 1894 of *Lecture Notes in Computer Science*, pages 306–319, Singapore, Sept. 2000. Springer-Verlag.

[14] Mozart Consortium. The Mozart programming system, 1999. Available from `www.mozart-oz.org`.

[15] J.-F. Puget. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, pages 359–366, Madison, WI, USA, July 1998. AAAI Press/The MIT Press.

[16] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelth National Conference on Artificial Intelligence*, volume 1, pages 362–367, Seattle, WA, USA, 1994. AAAI Press.

[17] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. The MIT Press, 1989.

[18] P. Van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, Cambridge, MA, USA, 1999.